Erik Mitchell
ID: 00001581305
Rank: 19, Score: 0.7367

Program 1 Report

Introduction

The applications of neural networks are vast, including many fields. With the neural networks we have learned in class, we can help make large advancements in these fields, such as in biotechnology. In this program, we are tasked with finding a connection between the sequence of amino acids, known as peptides, and whether they can penetrate the biofilm of a microorganism, which if penetrated can prevent diseases. We are to accomplish this task by using a feed-forward neural network, the numpy library, and the techniques learned from lecture.

Data Analysis

We are given two datasets, train.dat and test.dat. Each of these consist of peptide sequences, one per line, and a number, 1 or -1, that indicates whether they are anti-biofilm or not anti-biofilm. Here is an example of the data format:

```
1	DVELDLVEISPNALP
```

Train.dat has 1566 peptide sequences with labels. Test.dat has 392 peptide sequences without any labels. The dataset within train.dat is imbalanced, meaning that there is an unproportionate amount of -1 samples compared to 1 samples. We will address this during our data manipulation.

Approach

Data Manipulation

1. *Data Encoding*

   I decided to use kmers to develop substrings of the peptides, capturing the amino_acid sequences. I generated the kmers for k=3, k=2, and k=1. With these iterative kmers built, I generated a mapping for them to be placed on a matrix, translating the counts of kmers into an encoded matrix. With this technique, the features of the peptides are translated into a numeric form, while still containing information on the sequence of amino acids (via the kmers).

2. *Data Balancing & Shuffling*

As introduced in the data analysis section, the given training dataset has an uneven amount of antibiofilm and non-antibiofilm samples. Off-balanced datasets can cause models to overfit to a particular type of sample, making them unable to generalize to different data inputs. In order to balance the dataset, I made copies of the unrepresented samples. The ordering of the data in the dataset can also highly influence the performance of the model. It's important that there is no correlation between the order of the data within the dataset and the type of data it represents. As such, I ensured that the dataset was shuffled before training the model.

Model Definition

The feed-forward neural network contains 4 layers each followed by an activation function. The input layer has 436 neurons, which represents the number of features extracted from the peptides using k=3 for the iterative k-mers. The hidden layers have 64, 32, and 16 neurons respectively, with the output layer having 1 neuron. With a large number of input features, I found it important to have multiple hidden layers so that the model could accurately capture and learn the dense information. I used the tanh activation function since its range is between 1 and -1, which matches the labels for the samples. The MSE loss function is used for its simplicity and for its smooth differentiability. I used a learning rate of learning rate of 0.01 with 210 epochs. These parameters seemed to give the best results without overfitting.

Results

Numpy Neural Network

The model trains very well on the training dataset, with the loss dramatically dropping during the first epochs and remaining low. The MCC, calculated on the training validation set, is very good at .95. This MCC score is lower than what is shown in the CLP, indicating that there may be some overfitting to the training set.

```
MCC:  0.9563547080375059
```

```
Epoch 0, Loss 0.27323106
Epoch 20, Loss 0.00020241
Epoch 40, Loss 0.00007459
Epoch 60, Loss 0.00004519
Epoch 80, Loss 0.00003221
Epoch 100, Loss 0.00002493
Epoch 120, Loss 0.00002028
Epoch 140, Loss 0.00001707
Epoch 160, Loss 0.00001472
Epoch 180, Loss 0.00001292
Epoch 200, Loss 0.00001151
Epoch 220, Loss 0.00001037
Epoch 240, Loss 0.00000943
```

Pytorch Neural Network

Unfortunately, I was unable to replicate the results on the Pytorch model. I had some difficulties replicating the logic I implemented to the Pytorch modules, and as such the results are not great at all. The loss does not drop as much as it should, stagnating at 0.07.

```
MCC:  0.4163026153212167
```

```
Epoch 0, Loss 0.40114225
Epoch 20, Loss 0.16741812
Epoch 40, Loss 0.15037435
Epoch 60, Loss 0.13609270
Epoch 80, Loss 0.11664521
Epoch 100, Loss 0.10003806
Epoch 120, Loss 0.09154387
Epoch 140, Loss 0.08565791
Epoch 160, Loss 0.08253271
Epoch 180, Loss 0.07942265
Epoch 200, Loss 0.07709282
Epoch 220, Loss 0.07573644
Epoch 240, Loss 0.07438206
```

Network Comparison

Interestingly, I found it easier to implement my own internal logic for the neural network, using the lectures and examples as a guide, rather than implementing that same logic into Pytorch. Admittedly, I do not have much experience with Pytorch, so it was definitely a learning experience for me. In the future, I think I would instead follow the Pytorch standards more closely, rather than trying to replicate the logic of a different model so that I can achieve better results.

Conclusion

Overall, I was able to train a feed-forward neural network to make fairly accurate predictions of whether a given peptide is antibiofilm or not. I learned and implemented encoding techniques, manipulated data, and designed a neural network, all core principles that can be applied to any deep learning project. With these takeaways, I hope to build on my learning to solve more complex problems.